Parsing Context-Free Languages
Alon Lavie
Language Technologies Institute
Carnegie Mellon University

Malta
November 2009

Reading:
Jurafsky and Martin,
“Speech and Language Processing”
Chapter 10
Parsing Algorithms

- CFGs are basis for describing (syntactic) structure of NL sentences
- Thus - Parsing Algorithms are core of NL analysis systems
- Recognition vs. Parsing:
  - *Recognition* - deciding the membership in the language:
    For a given grammar $G$, an algorithm that given an input $w$ decides: is $w \in L(G)$?
  - *Parsing* - Recognition + producing a parse tree for $w$
- Is parsing more “difficult” than recognition? (time complexity)
- Ambiguity - a parse for $w$ or all parses for $w$?
  - Identifying the “correct” parse
  - Ambiguity representation - an input may have exponentially many parses
Parsing Algorithms

Parsing General CFLs vs. Limited Forms

• Efficiency:
  – Deterministic (LR) languages can be parsed in linear time
  – A number of parsing algorithms for general CFLs require \( O(n^3) \) time
  – Asymptotically best parsing algorithm for general CFLs requires \( O(n^{2.376}) \), but is not practical

• Utility - why parse general grammars and not just CNF?
  – Grammar intended to reflect actual structure of language
  – Conversion to CNF completely destroys the parse structure
Top-Down vs. Bottom-Up Parsing

Top-Down Parsing:

- Construct the parse-tree starting from the root (“S”) of the grammar
- At each step, expand a non-terminal using one selected grammar rule
- match terminal nodes with the input
- backtrack when tree is inconsistent with input
- Advantage: only constructs partial trees that can be derived from the root “S”
- Problems: efficiency, handling ambiguity, left-recursion

Bottom-Up Parsing:

- Construct a parse starting from the input symbols
- Build constituents from sub-constituents
- When all constituents on the RHS of a rule are matched, create a constituent for the LHS of the rule
- Advantage: only creates constituents that are consistent with the input
- Problems: efficiency, handling ambiguity
Top-Down vs. Bottom-Up Parsing

- Various CFG parsing algorithms are a hybrid of Top-Down and Bottom-Up
- Attempt to combine the advantages of both
- A *Chart* allows storing partial analyses, so that they can be shared or memorized
- Ambiguity Packing allows efficient storage of ambiguous analyses
The Earley Parsing Algorithm

General Principles:

- A clever hybrid *Bottom-Up* and *Top-Down* approach
- *Bottom-Up* parsing completely guided by *Top-Down* predictions
- Maintains sets of “dotted” grammar rules that:
  - Reflect what the parser has “seen” so far
  - Explicitly predict the rules and constituents that will combine into a complete parse
- Time Complexity $O(n^3)$, but better on particular sub-classes
- First efficient parsing algorithm for general context-free grammars.
The Earley Parsing Method

- Main Data Structure: The “state” (or “item”)
- A state is a “dotted” rule and starting position:
  \[ A \rightarrow X_1 \ldots \bullet C \ldots X_m, p_i \]
- The algorithm maintains sets of states, one set for each position in the input string (starting from 0)
- We denote the set of states for position \( i \) by \( S_i \)
The Earley Parsing Algorithm

Three Main Operations:

- **Predictor:** If state \([A \rightarrow X_1 \ldots \bullet C \ldots X_m, j] \in S_i\) then for every rule of the form \(C \rightarrow Y_1 \ldots Y_k\), add to \(S_i\) the state \([C \rightarrow \bullet Y_1 \ldots Y_k, i]\).

- **Completer:** If state \([A \rightarrow X_1 \ldots X_m \bullet, j] \in S_i\) then for every state in \(S_j\) of form \([B \rightarrow X_1 \ldots \bullet A \ldots X_k, l]\), add to \(S_i\) the state \([B \rightarrow X_1 \ldots A \bullet \ldots X_k, l]\).

- **Scanner:** If state \([A \rightarrow X_1 \ldots \bullet a \ldots X_m, j] \in S_i\) and the next input word is \(x_{i+1} = a\), then add to \(S_{i+1}\) the state \([A \rightarrow X_1 \ldots a \bullet \ldots X_m, j]\).
The Earley Recognition Algorithm

- Simplified version with no lookaheads and for grammars without epsilon-rules
- Assumes input is string of grammar terminal symbols
- We extend the grammar with a new rule $S' \rightarrow S \$$
- The algorithm sequentially constructs the sets $S_i$ for $0 \leq i \leq n + 1$
- We initialize the set $S_0$ with $S_0 = \{[S' \rightarrow \bullet S \$, 0]\}
The Earley Recognition Algorithm

The Main Algorithm: parsing input $x = x_1...x_n$

1. $S_0 = \{[S' \rightarrow \bullet S \$, 0]\}$

2. For $0 \leq i \leq n$ do:
   - Process each item $s \in S_i$ in order by applying to it the *single* applicable operation among:
     - (a) Predictor (adds new items to $S_i$)
     - (b) Completer (adds new items to $S_i$)
     - (c) Scanner (adds new items to $S_{i+1}$)

3. If $S_{i+1} = \emptyset$, *Reject* the input

4. If $i = n$ and $S_{n+1} = \{[S' \rightarrow S \bullet$, 0]\} then *Accept* the input
Earley Recognition - Example

The Grammar:

(1) \( S \rightarrow NP \ V P \)
(2) \( NP \rightarrow art \ adj \ n \)
(3) \( NP \rightarrow art \ n \)
(4) \( NP \rightarrow adj \ n \)
(5) \( VP \rightarrow aux \ VP \)
(6) \( VP \rightarrow v \ NP \)

The original input: “\( x = \) The large can can hold the water”
POS assigned input: “\( x = art \ adj \ n \ aux \ v \ art \ n \)”
Parser input: “\( x = art \ adj \ n \ aux \ v \ art \ n \$””
Earley Recognition - Example

The input: “$x = \text{art} \ \text{adj} \ n \ \text{aux} \ v \ \text{art} \ n \ \$”

$S_0$: $[S' \rightarrow \bullet S \ \$, \ 0]$
$[S \rightarrow \bullet NP \ VP \ , \ 0]$
$[NP \rightarrow \bullet art \ adj \ n \ , \ 0]$
$[NP \rightarrow \bullet art \ n \ , \ 0]$
$[NP \rightarrow \bullet adj \ n \ , \ 0]$

$S_1$: $[NP \rightarrow \text{art} \ \bullet \ adj \ n \ , \ 0]$
$[NP \rightarrow \text{art} \ \bullet \ n \ , \ 0]$
Earley Recognition - Example

The input: “$x = \text{art adj} \ n \ \text{aux} \ v \ \text{art} \ n \ \$$”

$S_1$: $[NP \rightarrow \text{art} \bullet \text{adj} \ n \ , \ 0]$

$[NP \rightarrow \text{art} \bullet \ n \ , \ 0]$

$S_2$: $[NP \rightarrow \text{art adj} \bullet \ n \ , \ 0]$
Earley Recognition - Example

The input: “$x = \text{art adj } n \text{ aux v art n }$”

\(S_2:\ [NP \rightarrow \text{art adj } \bullet n , 0]\)

\(S_3:\ [NP \rightarrow \text{art adj } n \bullet , 0]\)
Earley Recognition - Example

The input: “\(x = \text{art adj n aux v art n }\) $”

\[S_3: \begin{aligned}
[NP & \rightarrow \text{art adj n } \bullet, \ 0] \\
[S & \rightarrow NP \ \bullet VP, \ 0] \\
[VP & \rightarrow aux VP, \ 3] \\
[VP & \rightarrow v NP, \ 3]
\end{aligned}\]

\[S_4: \begin{aligned}
[VP & \rightarrow aux \ \bullet VP, \ 3]
\end{aligned}\]
Earley Recognition - Example

The input: “x = art adj n aux v art n $”

\[ S_4: \left[ VP \rightarrow aux \bullet VP , 3 \right] \]
\[ \left[ VP \rightarrow aux VP , 4 \right] \]
\[ \left[ VP \rightarrow v NP , 4 \right] \]

\[ S_5: \left[ VP \rightarrow v \bullet NP , 4 \right] \]
Earley Recognition - Example

The input: "x = art adj n aux v art n $"

\[S_5:\]

\[VP \rightarrow v \bullet NP , 4\]
\[NP \rightarrow \bullet art \ adj n , 5\]
\[NP \rightarrow \bullet art n , 5\]
\[NP \rightarrow \bullet adj n , 5\]

\[S_6:\]

\[NP \rightarrow art \bullet adj n , 5\]
\[NP \rightarrow art \bullet n , 5\]
Earley Recognition - Example

The input: “x = art adj n aux v art n $”

\[ S_6: \ [NP \rightarrow \text{art} \bullet \text{adj n} , 5] \]
\[ [NP \rightarrow \text{art} \bullet n , 5] \]

\[ S_7: \ [NP \rightarrow \text{art} n \bullet , 5] \]
Earley Recognition - Example

The input: “$x = \text{art adj n aux v art n}$”

\[\begin{align*}
S_7: & \quad [NP \rightarrow \text{art n} \bullet, 5] \\
& \quad [VP \rightarrow v NP \bullet, 4] \\
& \quad [VP \rightarrow \text{aux } VP \bullet, 3] \\
& \quad [S \rightarrow NP VP \bullet, 0] \\
& \quad [S' \rightarrow S \bullet $, 0]
\end{align*}\]

\[\begin{align*}
S_8: & \quad [S' \rightarrow S $ \bullet, 0]
\end{align*}\]
Parsing with an Earley Parser

• We need to keep back-pointers to the constituents that we combine together when we complete a rule

• Each item must be extended to have the form
  \[ A \rightarrow X_1(pt_1)... \bullet C...X_m,j \], where the \( pt_i \) are “pointers” to the already found RHS sub-constituents

• the constituents and the pointers can be created during Scanner and Completer

• At the end - reconstruct parse from the “back-pointers”
Earley Parsing - Example

The input: “$x = \text{art adj n aux v art n}$”
Earley Parsing - Example

The input: “x = art adj n aux v art n $”

\[ S_0: \quad [S' \rightarrow \bullet S \$, 0] \]
\[ [S \rightarrow \bullet NP \ VP , 0] \]
\[ [NP \rightarrow \bullet art \ adj \ n , 0] \]
\[ [NP \rightarrow \bullet art \ n , 0] \]
\[ [NP \rightarrow \bullet adj \ n , 0] \]

\[ S_1: \quad [NP \rightarrow art_1 \ \bullet adj \ n , 0] \quad 1 \quad art \ (0,1) \]
\[ [NP \rightarrow art_1 \ \bullet n , 0] \]
Earley Parsing - Example

The input: “x = art adj n aux v art n $”

$S_1$: $[NP \rightarrow art_1 \bullet adj n , 0]$

$[NP \rightarrow art_1 \bullet n , 0]$

$S_2$: $[NP \rightarrow art_1 adj_2 \bullet n , 0]$ 2 adj (1,2)
Earley Parsing - Example

The input: “x = art adj n aux v art n $”

\[ S_2: [NP \rightarrow art_1 \ adj_2 \ \bullet n, 0] \]

\[ S_3: [NP_4 \rightarrow art_1 \ adj_2 \ n_3 \ \bullet, 0] \]

3 \ n (2,3)

4 \ NP \rightarrow art_1 \ adj_2 \ n_3 (0,3)
Earley Parsing - Example

The input: “$x = \text{art} \ \text{adj} \ n \ \text{aux} \ v \ \text{art} \ n \ \$”

$S_3$:  
$[NP_4 \rightarrow \text{art}_1 \ \text{adj}_2 \ n_3 \bullet, \ 0]$
$[S \rightarrow NP_4 \bullet VP, \ 0]$
$[VP \rightarrow \bullet aux \ VP, \ 3]$
$[VP \rightarrow \bullet v \ NP, \ 3]$

$S_4$:  
$[VP \rightarrow aux_5 \bullet VP, \ 3]$

5 aux (3,4)
Earley Parsing - Example

The input: “x = art adj n aux v art n $”

$S_4$: \[ VP \rightarrow aux_5 \bullet VP , 3 \]
\[ VP \rightarrow \bullet aux VP , 4 \]
\[ VP \rightarrow \bullet v NP , 4 \]

$S_5$: \[ VP \rightarrow v_6 \bullet NP , 4 \]
6 \( v \ (4,5) \)
The input: “$x = \text{art adj n aux v art n }$”

$S_5$:  

\[ VP \rightarrow v_6 \cdot NP , 4 \]
\[ NP \rightarrow \text{art adj n} , 5 \]
\[ NP \rightarrow \text{art n} , 5 \]
\[ NP \rightarrow \text{adj n} , 5 \]

$S_6$:  

\[ NP \rightarrow \text{art}_7 \cdot \text{adj n} , 5 \] 7 art (5,6)
\[ NP \rightarrow \text{art}_7 \cdot n , 5 \]
Earley Parsing - Example

The input: “$x = \text{art adj n aux v art n}$”

$S_6$:  
\[ NP \rightarrow \text{art}_7 \bullet \text{adj n , 5} \]
\[ NP \rightarrow \text{art}_7 \bullet n , 5 \]

$S_7$:  
\[ NP_9 \rightarrow \text{art}_7 n_8 \bullet , 5 \]
\[ n (6,7) \]
\[ NP \rightarrow \text{art}_7 n_8 (5,7) \]
Earley Parsing - Example

The input: “$x = \text{art adj n aux v art n}$”

\[ S_7: \begin{align*}
[NP_9 & \rightarrow \text{art}_7 \text{ n}_8 \bullet , 5] \\
[VP_{10} & \rightarrow \text{v}_6 \ NP_9 \bullet , 4] \\
[VP_{11} & \rightarrow \text{aux}_5 \ VP_{10} \bullet , 3] \\
[S_{12} & \rightarrow NP_4 \ VP_{11} \bullet , 0] \\
[S' & \rightarrow S \bullet \$ , 0]
\end{align*} \]

\[ S_8: \begin{align*}
[S' & \rightarrow S \$ \bullet , 0]
\end{align*} \]
Efficient Representation of Ambiguities

- a Local Ambiguity - multiple ways to derive the same substring from a non-terminal $A$

- What do local ambiguities look like with Earley Parsing?
  - Multiple items in the constituent chart of the form
    \[ A \to X_1(pt_1)...X_m(pt_m)](p_k,p_j), \text{ with the same } A, p_j \text{ and } p_k.\]

- Local Ambiguity Packing: create a single item in the Chart for $A(p_j,p_k)$, with pointers to the various possible derivations.

- $A(p_j,p_k)$ can then be a sufficient “back-pointer” in the chart

- Allows to efficiently represent a very large number of ambiguities (even exponentially many)

- Unpacking - producing one or more of the packed parse trees by following the back-pointers.
Time Complexity of Earley Algorithm

- Algorithm iterates for each word of input (i.e. $n$ iterations)
- How many items can be created and processed in $S_i$?
  - Each item in $S_i$ has the form $[A \rightarrow X_1 \ldots \bullet C \ldots X_m, j]$, $0 \leq j \leq i$
  - Thus $O(n)$ items
- The Scanner and Predictor operations on an item each require constant time
- The Completer operation on an item adds items of form $[B \rightarrow X_1 \ldots A \bullet \ldots X_k, l]$ to $S_i$, with $0 \leq l \leq i$, so it may require up to $O(n)$ time for each processed item
- Time required for each iteration ($S_i$) is thus $O(n^2)$
- Time bound on entire algorithm is therefore $O(n^3)$
Time Complexity of Earley Algorithm

Special Cases:

- *Completer* is the operation that may require $O(i^2)$ time in iteration $i$
- For unambiguous grammars, Earley shows that the completer operation will require at most $O(i)$ time
- Thus time complexity for unambiguous grammars is $O(n^2)$
- For some grammars, the number of items in each $S_i$ is bounded by a constant
- These are called *bounded-state* grammars and include even some ambiguous grammars.
- For bounded-state grammars, the time complexity of the algorithm is linear - $O(n)$